An Architecture For Web Deployment Of Decision Support Systems Based On Probabilistic Graphical Models With Applications

Madsen, Anders Læsø; Karlsen, Martin ; Barker, Gary C.; Garcia, Ana Belen; Hoorfar, Jeffrey ; Jensen, Frank

Publication date:
2012

Document Version
Early version, also known as pre-print

An Architecture For Web Deployment Of Decision Support Systems Based On Probabilistic Graphical Models With Applications

Tech Report TR-12-001
Publisher: Department of Computer Science, Aalborg University
ISBN: 1601-0590

Anders L. Madsen

HUGIN EXPERT A/S,

Gasværksvej 5, DK-9000 Aalborg, Denmark

and

Department of Computer Science,

Aalborg University, Selma Lagerlöfs Vej 300, DK-9220 Aalborg Ø, Denmark

Martin Karlsen

HUGIN EXPERT A/S,

Gasværksvej 5, DK-9000 Aalborg, Denmark

Gary C. Barker

Institute of Food Research,

Norwich Research Park, Colney, Norwich, NR4 7UA, United Kingdom

Ana Belen Garcia, Jeffrey Hoorfar

Technical University of Denmark,

National Food Institute, Mørkhøj Bygade 19, DK-2860 Søborg, Denmark

Frank Jensen

HUGIN EXPERT A/S,

Gasværksvej 5, DK-9000 Aalborg, Denmark

Håkan Vigre

Technical University of Denmark,

National Food Institute, Mørkhøj Bygade 19, DK-2860 Søborg, Denmark

December 21, 2012

This paper presents a web service-based architecture for deployment of decision support systems based on probabilistic graphical models and two applications of the architecture. The architecture has been developed to meet a unsatisfied customer need of being able to engage stakeholders at different levels in the knowledge engineering process and to efficiently deploy decision support systems based on probabilistic graphical models on the internet. This serves a number of purposes such as to give broader access to decision support based on probabilistic graphical models, to ensure involvement from stakeholders in the knowledge engineering, model and interface development process and to demonstrate the potential of probabilistic graphical models to a larger audience.

The architecture has been developed on top of the HUGIN tool. The HUGIN tool is a commercial-off-the-shelf (COTS) software package for construction, maintenance and deployment of probabilistic graphical models. The HUGIN software package consists of the HUGIN Graphical User Interface (HUGIN GUI) and the HUGIN Decision Engine (HDE).

The paper describes how the architecture has been used to develop and deploy two real-world models for decision support in food safety. One application relates to decision support on campylobacter vaccination of poultry in Denmark while the other application relates to quantification of hazards that arise from Staphylococcus aureus in milk sold as pasteurized in the United Kingdom. In both cases (and others) the architecture presented in this paper has served as a strong platform for the knowledge engineering process and for development and deployment of the underlying probabilistic models as effective decision support systems.

The architecture is an appropriate and efficient tool for easy deployment of probabilistic graphical models and fast development of prototype systems. It helps to ensure stakeholder involvement at an early point in the knowledge engineering and development process.

1 Introduction

Probabilistic Graphical Models (PGMs) such as, for instance, Bayesian networks and influence diagrams are increasingly popular knowledge representations for model-based decision support under uncertainty [40, 7, 37, 22, 26, 8, 27]. The availability of efficient

tools for construction and inference is a major reason for the success of PGMs. PGMs are being applied in a range of diverse domains including, but not limited to, finance, agriculture, food safety and bioinformatics [10, 41, 4, 37, 28].

PGMs provide a range of tools and methods for knowledge elicitation, knowledge representation and inference in highly structured domains with inherent uncertainty. A PGM is an ideal knowledge representation of problem domains where relationships between entities can be defined using (conditional) dependence and independence properties and where the strengths of relations can be defined using (conditional) probability distributions.

A PGM is an efficient knowledge representation that supports the integration of knowledge from diverse sources such as information from literature, experience and knowledge from subject matter experts and historical data. As a knowledge representation a PGM supports both belief update and decision making under uncertainty. A significant number of research papers on applying PGMs to support decision making under uncertainty has been produced since belief update in Bayesian network became feasible in the late 1980s, e.g., [31].

Despite successful applications of PGMs in a range of different domains, developing (new) applications using the technology often involves a not insignificant level of innovation and research. According to our experience a reason why many research applications of PGMs do not make it into commercial use is a perceived gap between standard tools for developing PGMs and End-User applications and lack of stakeholder involvement at an early stage. Due to technical and sophisticated user interfaces of standard tools for PGMs, it can be difficult for knowledge engineers to engage stakeholders at different levels and convince decision makers that the development, deployment and integration of decision support based on PGMs is simple and efficient.

A main motivation for the development efforts described in this paper was the lack of tools to support efficient and effective deployment of PGMs for decision support under uncertainty for commercial use, for demonstration purposes and as a means of engaging stakeholders at different levels in the system development process, i.e., engaging stakeholders in the model development process and in the design of model interfaces as early on in the process as possible. According to our experience it can be difficult for decision makers and End-Users to appreciate and understand the opportunities offered by the technology without a powerful, efficient and effective front-end to the underlying model. Furthermore, a front-end to the underlying model can help to ensure engagement and commitment to the knowledge engineering process from the subject matter experts as well as improve their understanding of knowledge elicitation.

From the beginning of the development process it was a clear objective that the infrastructure should support both fast deployment of PGMs as well as efficient and effective deployment of PGMs on the Internet. It should be possible to construct simple and advanced End-User interfaces to highly complex PGMs models with as few efforts as possible. The aim was to develop an infrastructure that would make it possible for knowledge engineers to fast and efficiently create front-ends to PGM models enabling End-Users to interact with the models using a language and framework they would be familiar with. The infrastructure developed is a platform for easy deployment of both

prototype and commercial quality systems and for integration into new and existing systems. The infrastructure has both low-level and high-level components to construct efficient interfaces.

The paper is organised as follows. Section 2 gives a short introduction to PGMs and the HUGIN Decision Engine. Section 3 presents in details the web service-based architecture for deployment of PGMs while Section 4 describes two different real-world use cases using the web architecture. Section 5 presents a discussion on the main findings and concluding remarks while Section 6 describes future work.

2 Preliminaries

This section gives a short introduction to PGMs and the HUGIN software tools.

2.1 Probabilistic Graphical Models

A PGM (in this paper a Bayesian network or influence diagram) over a set of variables consists of a qualitative and a quantitative part. The qualitative part – specified as an acyclic, directed graph (DAG) – encodes a set of dependence and independence relations over the variables in the model. In a Bayesian network the qualitative part describes dependence and independence relations over chance variables whereas in an influence diagram the qualitative part describes dependence and independence relations over chance variables conditional on decision variables (or nodes), information orderings and preference relations over chance and decision variables.

A Bayesian network $\mathcal{N} = (\mathcal{X}, \mathcal{G}, \mathcal{P})$ consists of a DAG $\mathcal{G} = (V, E)$ where V is the set of vertices and E is the set of directed edges and a set of probability distributions $\mathcal{P}$ where there is a one-to-one correspondence between variables $\mathcal{X}$ and vertices V. For each variable $X \in \mathcal{X}$ there is a conditional probability distribution $P(X \,|\, \mathrm{pa}(X)) \in \mathcal{P}$. $\mathcal{N}$ is an efficient representation of a joint probability distribution $P(\mathcal{X})$ over $\mathcal{X}$ when $\mathcal{G} = (V, E)$ is not dense. The joint probability distribution $P(\mathcal{X})$ represented by $\mathcal{N}$ factorizes according to the structure of $\mathcal{G}$ as

$$P(\mathcal{X}) = \prod_{X \in \mathcal{X}} P(X \,|\, \mathrm{pa}(X)). \tag{1}$$

Belief update in a Bayesian network $\mathcal{N} = (\mathcal{X}, \mathcal{G}, \mathcal{P})$ is the process of computing the posterior marginal $P(Y \,|\, \epsilon)$ for each variable $Y \in \mathcal{X}$ given the evidence ϵ. In principle, the posterior marginal distribution of Y given evidence $\epsilon = \{\epsilon_1, \ldots, \epsilon_n\}$ is computed as

$$P(Y \,|\, \epsilon) \propto \sum_{X \in \mathcal{X} \setminus \{Y\}} \prod_{X \in \mathcal{X}} P(X \,|\, \mathrm{pa}(X)) \prod_{\epsilon_i \in \epsilon} f(\epsilon_i), \tag{2}$$

where $f(\epsilon_i)$ is a finding potential enforcing ϵ_i where ϵ is assumed to consist of a set of variable instantiations.

An (Perfect Recall) Influence Diagram (PRID) is, in principle, a Bayesian network augmented with facilities for supporting decision making under uncertainty under the

assumption of regularity (a total order on the decisions) and no-forgetting (all past observations and decisions are recalled by the decision maker) [17]. A Limited Memory Influence Diagrams (LIMIDs) is an influence diagram where the regularity and no-forgetting assumptions have been lifted [30]. A PRID is a special case of the LIMID.

A LIMID $\mathcal{N} = (\mathcal{X}, \mathcal{G}, \mathcal{P}, \mathcal{U})$ consists of a DAG $\mathcal{G} = (V, E)$, a set of conditional probability distributions $\mathcal{P}$ and a set of (local) utility functions $\mathcal{U}$. The variables of $\mathcal{N}$ are partitioned into the set of chance variables $\mathcal{X}_C$ and the set of decision variables $\mathcal{X}_D$. The utility functions of $\mathcal{N}$ are represented as utility nodes V_U in $\mathcal{G}$. For each variable $X \in \mathcal{X}_C$ there is a conditional probability distribution $P(X \,|\, \mathrm{pa}(X))$ and for each utility node $U \in V_U$ there is a local utility function $u(\mathrm{pa}(U))$, whereas a decision variable $D \in \mathcal{X}_D$ does not have any associated function or distribution. Instead a decision variable $D \in \mathcal{X}_D$ has a policy δ. The expected utility function $EU(\mathcal{X})$ represented by $\mathcal{N}$ factorizes according to the structure of $\mathcal{G}$ as

$$EU(\mathcal{X}) = \prod_{X \in \mathcal{X}_C} P(X \,|\, \mathrm{pa}(X)) \sum_{U \in V_U} u(\mathrm{pa}(U)). \tag{3}$$

The task of solving an influence diagram is to find a strategy $\Delta = \{\delta_1, \ldots, \delta_m\}$ consisting of one policy for each decision $D \in V_D$. A policy is a mapping from (relevant) past observations and decisions to alternatives available for the decision under consideration.

The solution to an influence diagram or a LIMID is a (optimal) strategy $\Delta = (\delta_1, \ldots, \delta_m)$ consisting of a policy δ_i for each decision $D_i \in \{D_1, \ldots, D_m\}$ and the expected utility of adhering to Δ. By encoding each policy δ_i as a conditional probability distribution $P(D_i \mid \mathrm{pa}(D_i))$, it is possible to compute the probability of future decisions and observations (conditional on the policy Δ) from the joint probability distribution [39]

$$P(\mathcal{X} \,|\, \Delta) = \prod_{X \in \mathcal{X}_C} P(X \,|\, \mathrm{pa}(X)) * \prod_{i=1}^{m} P(D_i \,|\, \mathrm{pa}(D_i)). \tag{4}$$

For ease of presentation this paper considers PGMs with discrete variables only.

2.2 Example

SimpleTrace is a PGM developed at the Institute of Food Research, United Kingdom, as an example to illustrate the role of domain knowledge in source level inference and to highlight the role of information uncertainty in decision making. The SimpleTrace example mimics a typical food safety scenario that involves a chain of steps and a developing population[1].

SimpleTrace allows reasoning even when a considerable amount of uncertainty is associated with the information that quantifies steps in the food chain. It includes a growth process and a potential source at each step along the chain with prior probabilities describing sources and conditional probability distributions describing dependencies.

Figure 1 shows the structure of SimpleTrace. It represents a finite chain of events with five sequential (observable) elements numbered as $i = 0, \ldots, 4$. At each stage of

[1] http://bbn.ifr.ac.uk/btmodeller/index.php/SimpleTrace

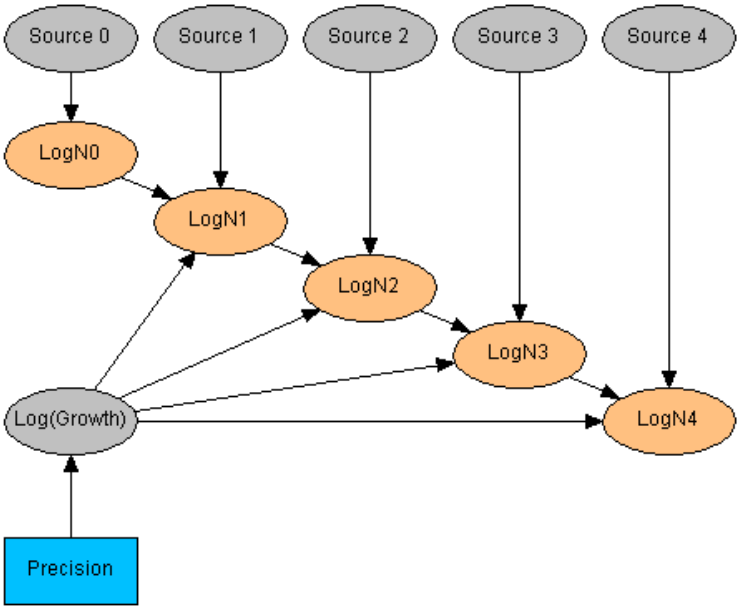

Figure 1: The SimpleTrace model.

the chain an agent has a concentration N_i; the representative variable is the logarithm $\log N_i$. At each stage there is a possible source of the agent with a fixed concentration 0.1. Boolean variables (Source i) quantify the uncertain sources and all sources have prior probability 0.005.

Between consecutive elements in the chain finite concentrations of the agent grow with an uncertain growth factor, in the range 1 to 100; the growth factor determines the population dynamics expressed as $\log(N_{i+1}/N_i) \sim \text{Beta}(m, m, 0, 2)$. A decision node (Precision) can be used to choose between high ($m = 200$), medium ($m = 20$) and low ($m = 1$) precision concerning the growth factor.

SimpleTrace illustrates a *biotracing* process. Observations of the end point concentration, i.e., evidence entered at $logN_4$, provides posterior belief concerning the sources. The ability to identify a source as the origin of observed agents (i.e., *biotrace*) depends on the precision assigned to knowledge of the growth factor.

Figure 2 shows the result of entering and propagating evidence that includes high precision concerning the growth factor and specific end point observation $\log N_4 = 1.35$, i.e., $\log N_4$ in the interval from 1.25 to 1.5. It is clear from the figure that Source 2 is the most likely source (cause) of the observed contamination since $P(\text{Source } 2 \mid \epsilon = \{\log N_4 = 1.35, \text{Precision} = \text{high}\}) = 0.9496$.

The SimpleTrace model, and particularly its implementation, is a powerful tool for the communication and demonstration of the biotracing concept. After witnessing the inference process, and the role of information uncertainty, many food safety experts can translate the principles into real food chain systems and so begin the development of dedicated diagnostic tools.

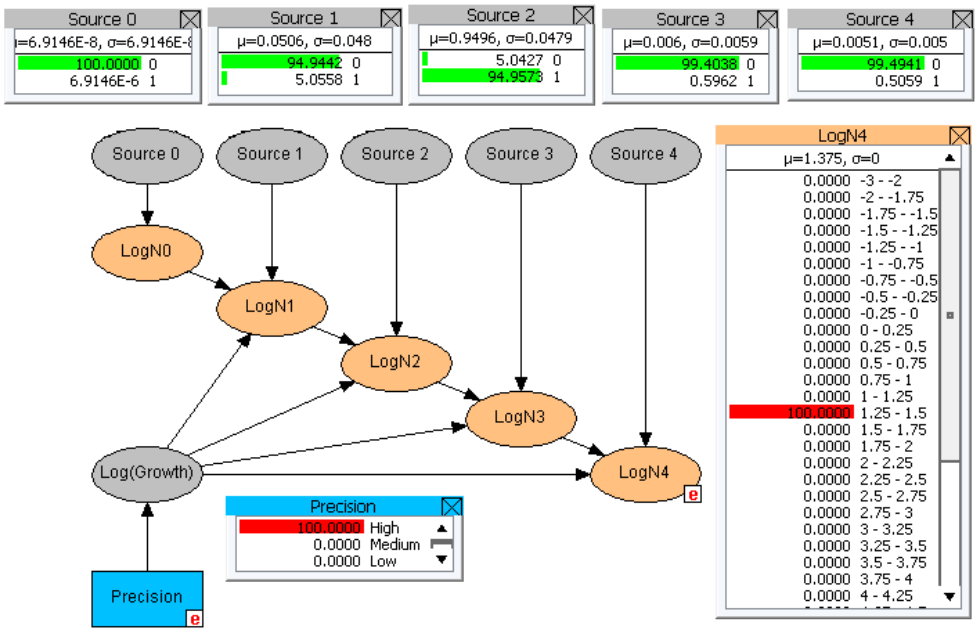

Figure 2: The SimpleTrace model in HUGIN GUI with evidence entered and propagated.

2.3 HUGIN Decision Engine

The HDE is a highly stable and efficient inference engine for PGMs. The original HUGIN Shell dates back to the late 1980s [2]. The functionality and efficiency of the HDE have been described in a number of papers [19, 35, 33].

The HDE is a general purpose inference engine for Bayesian networks and influence diagrams. It supports PGMs with discrete variables, mixed discrete and continuous variables (assuming conditional linear Gaussian distributions), LIMIDs, object-oriented models, learning from data of both structure and parameters etc. The inference algorithm for both belief update in Bayesian networks and solving LIMIDs is based on message passing in a junction tree structure using the HUGIN algorithm [21, 20, 23, 34, 29, 30]. A junction tree $\mathcal{T} = (\mathcal{C}, \mathcal{S})$ of, for instance, a Bayesian network $\mathcal{N} = (\mathcal{X}, \mathcal{G}, \mathcal{P})$ is constructed by moralization and triangulation of G where $\mathcal{C}$ are the cliques and $\mathcal{S}$ are the separators of $\mathcal{T}$ [7, 22, 26].

The HUGIN Decision Engine has Application Programming Interfaces (APIs) for C, C++, C#, Java and Visual Basic. The infrastructure referred to as the HUGIN Web Service API is described in detail in the next section supports deployment of models and is developed on top of the HDE using the HUGIN Java API.

3 Architecture

This section describes in detail the design and application of the HUGIN Web Service API on top of the HUGIN Decision Engine. The HUGIN Web Service API is an architecture for efficient and effective deployment of PGMs on the internet. It supports the knowledge elicitation process by enabling knowledge engineers and developers to deploy prototype and commercial quality models on the internet as discussed above.

3.1 Web Applications

A web application is an application that runs in a web browser. Application code is downloaded using the HyperText Transfer Protocol (HTTP) [11] and executed in the browser JavaScript engine, the graphical user interface is generated using HyperText Markup Language (HTML) and rendered by the browser. A network connected compatible browser is the sole requirement for using a web application, in effect making web applications cross-platform compatible, and cutting down deployment complexity as no client-side installation is required.

Most web applications use the client-server model, where a client (application code executing in browser JavaScript engine) interacts with one or more servers as application state progresses. A Web service is the method of communication between the client and server, typically using the HTTP protocol.

A Web application is usually a mashup of several different parts of functionality and Web services.

3.2 A framework for PGM based Web applications

To support rapid development of PGM based Web applications, a flexible framework for exercising the functionality of the HDE has been developed. The framework is a combination of a number of separate blocks of core functionality:

- **HUGIN Web Service API**. Basic server interaction, HDE API function invocations using HTTP. Objects live on server. Loading/saving data by use of third-party or custom build Web services.
- **JavaScript for HUGIN Web Service API**. Provide JavaScript hooks to the Web Service API such that the HDE can be scripted from a browser.
- **HUGIN Widgets Library**. A set of commonly used GUI items that developers can use in their Web applications. The widgets use JavaScript for the HUGIN Web Service API.

Representational State Transfer (REST) is a well-known and popular architectural style for web application and Web service development. The REST architectural style imposes *"a number of constraints that attempts to minimize latency and network communication while at the same time maximizing the independence and scalability of component implementations"* [12]. The REST principles were applied as a guideline in the design of the HUGIN Web Service HTTP based communication.

The diagram in Figure 3 illustrates the separate blocks of functionality in the HUGIN Web Service framework and the common points of integration with user code and third-party Web services.

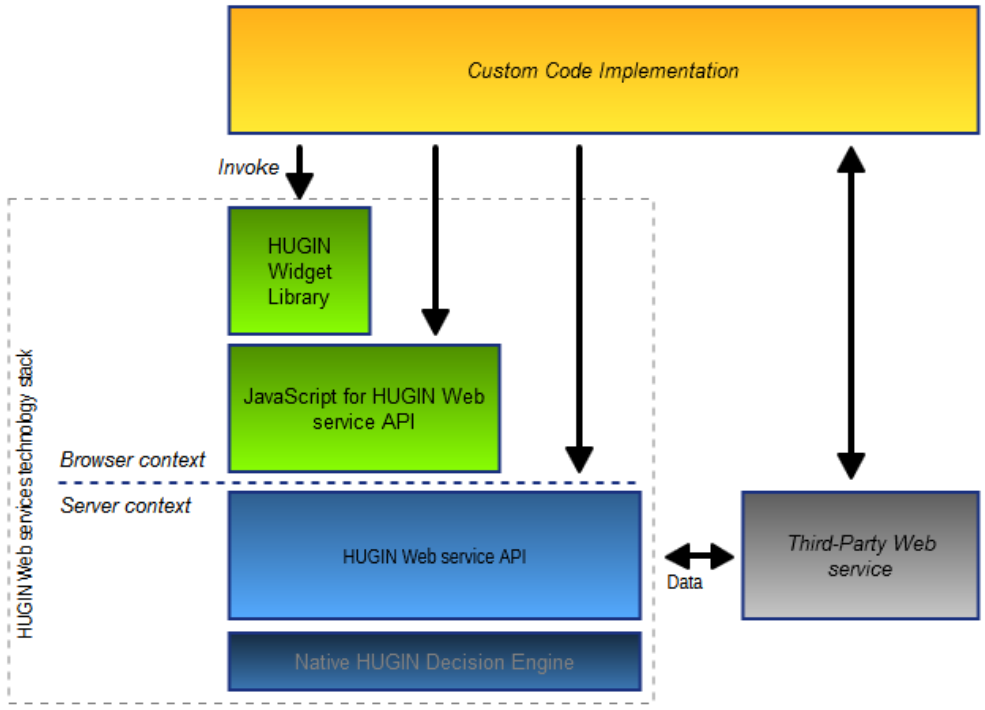

Figure 3: HUGIN Web Service technology stack.

The stack bottom represents the parts that live in a server context. This is the HUGIN Web Services API which is a server process implemented on the native HDE. Any communication to and from the Web service API is done using HTTP.

Directly above are the parts that live in a browser context. This is the JavaScript for HUGIN Web Service API that provides hooks to the browser JavaScript engine and abstracting away the induced HTTP communication to the Web service API. And the HUGIN Widgets library, a set of GUI building blocks that a developer can use for making buttons and other widgets for interacting with a PGM.

The top part represents the custom code implementation made by the developer. In the case of a browser based Web application, this code is executing in the browser JavaScript engine. A developer has the choice of communicating directly with the Web service API by means of HTTP, or by using the JavaScript for the HUGIN Web Service API and optionally exercising it through the HUGIN Widgets Library as well. Custom code can also be a script running in any scripting environment such as a server context PHP script (as demonstrated below) or a native application written in any programming language, in which case the only option for exercising the HDE is by direct HTTP communication.

Finally, any third-party Web service can be used for loading and storing data and interacting with the Web service API over HTTP.

3.3 Web Service API

To provide a HTTP interface the HDE has been integrated in a web server, the server process. The web server is based on a Java implementation on top of the regular HUGIN Java API. The popular Java based web server implementation Jetty [24] is used for providing basic asynchronous HTTP handling.

Following the REST principles, requests and responses are built around the transfer of representations of resources. Each resource is represented by a unique Uniform Resource Locator (URL) and is manipulated using a set of standard HTTP methods. This means that the server process considers any PGM related object to be a URL addressable resource. Book keeping logic for translating between internal object references and URLs has been developed.

All resources (the decision engine entry point, Domains, Nodes, Tables etc.) are exposed over HTTP as URLs formatted using a **/rest/{resource-type}/{ID}** like naming convention, where the type of a resource and its unique instance ID can be extracted from the URL. Resources are grouped hierarchically according to the relation of ownership between PGM objects. That is, a Domain has a number of Nodes, each Node has a Model, each Model has a Table and so forth. The location of any resource within this hierarchy can be inferred from the URL. The hierarchical ordering of resources is an effective construct for controlling memory allocation and de-allocation, as deleting a resource implicitly deletes the entire branch of resources below it, providing automatic clean up of orphaned PGM related objects whenever any object is deleted.

Examples of resource URLs:

- Basic entry point for the HUGIN Decision Engine

  `/rest`

- Domain instances

  `/rest/domain/{ID}`

- Node instances (always belongs to a specific Domain instance)

  `/rest/domain/{ID}/node/{ID}`

- Table instances (always belongs to a specific Domain instance)

  `/rest/domain/{ID}/table/{ID}`

- Model instances (always belongs to a specific Domain instance)

`/rest/domain/{ID}/model/{ID}`

- Clique instances (always belongs to a specific Domain instance)

  `/rest/domain/{ID}/clique/{ID}`

- Junction Tree instances (always belongs to a specific Domain instance)

  `/rest/domain/{ID}/junctiontree/{ID}`

Functions for manipulating an object are also addressable by an URL. Examples:

- construct a new Domain object

  `/rest/newDomain`

- to propagate findings in a Domain

  `/rest/domain/{ID}/propagate`

- to get the name of a Node

  `/rest/domain/{ID}/node/{ID}/getName`

Functions are invoked using a HTTP request with methods GET or POST. Adhering to the REST principles and the HTTP specification, GET requests are used for idempotent functions, POST requests are used for functions that have side effects (that results in a change in the state of the decision engine). For performance considerations, the client should cache and re-use the result of any GET requests until the next POST request has been submitted.

Parameters passed to any function are formatted as an URL encoded querystring with a set of one or more anonymous fields and values. The number of fields depends on the number of parameters that the function requires.

Examples of GET requests (some lines are broken for presentation purposes):

- Getting a node by name from a domain

```
GET /rest/domain/67ddd8e9-0105-4323-8804-614ada1a7bf4/
     getNodeByName?=A
Response: 200 OK
Response Body: /rest/domain/
                67ddd8e9-0105-4323-8804-614ada1a7bf4/node/
                25c0e5a9-16bd-423b-85a6-fd8e7cf67a45
```

- Get the belief for state 1 for a discrete node

```
GET /rest/domain/67ddd8e9-0105-4323-8804-614ada1a7bf4/node/
     25c0e5a9-16bd-423b-85a6-fd8e7cf67a45/getBelief?=1
Response: 200 OK
Response Body: 0.25
```

- Get the 10 first data items of a discrete node table

```
GET /rest/domain/67ddd8e9-0105-4323-8804-614ada1a7bf4/table/
     7cc8452f-7049-40df-afbb-4760796d99e1/getData?=0&=10
Response: 200 OK
Response Body: 0.2 0.5 0.1 0.4 0.6 0.2 0.2 0.3 0.9 0.1
```

Examples of POST requests:

- Propagate evidence in a domain

```
POST /rest/domain/67ddd8e9-0105-4323-8804-614ada1a7bf4/
      propagate
Request body: =sum&=normal
Response: 200 OK
```

- Select state 0 of a discrete chance node

```
POST /rest/domain/67ddd8e9-0105-4323-8804-614ada1a7bf4/node/
      25c0e5a9-16bd-423b-85a6-fd8e7cf67a45/selectState
Request Body: =0
Response: 200 OK
```

Due to the nature of HTTP based Web services the server may not be notified when a client no longer needs a resource. Therefore, the server process performs garbage collection of untouched domain resources at regular intervals. Any Domain resource that has not been touched — that is, have had any functions invoked on it or on any of the other resources that it owns — is deleted and the memory reclaimed.

Error conditions are communicated using the standard HTTP status codes as described below:

200 The operation was completed successfully.

400 An error occurred, this may be due to a number of things, e.g., pre-conditions for invoking a function was not met, wrong parameters, etc. More information about the particular error can be found by inspecting the HTTP response body.

404 This means that a resource is not found. Often this is because the URL is wrong or because the resource in question has already been deleted or garbage collected.

3.4 JavaScript for Web Service API

The JavaScript for Web service API provides JavaScript hooks to the Web service API such that the HDE can be scripted from a browser. The API is a wrapper for the functionality provided by the HUGIN Web Service API. It translates any function call into a proper HTTP request for exercising the remote decision engine. Errors are thrown appropriately when error conditions arise, perfectly in line with normal JavaScript error handling. Communication with the Web service API is performed using the browser provided XMLHttpRequest functionality to invoke a HTTP request.

The JavaScript for the Web service API is implemented entirely in JavaScript code. As JavaScript is a dynamic, weakly typed language, it is a challenge to verify the correctness of larger JavaScript libraries. Therefore, the library has been developed to conform to a stricter subset of JavaScript imposed by the tool JSLint. JSLint is a static code analysis tool that checks if the code complies with a number of coding rules [25].

The JavaScript for Web service API is provided in the form of a .js JavaScript file. To ensure a short library loading time, steps were taken to minimize the byte footprint of the .js file. This is a widely used procedure known as minification, where the goal is to preserve the operational qualities of the code while reducing its overall byte footprint. Minification of JavaScript code was performed using the YUI Compressor, which removes unneeded characters from the .js file, as well as shortens internal variable names and perform other optimizations that lead to a functionally equivalent library with smaller byte footprint [52].

The .js library file can be included in HTML using the regular script inclusion tags as:

```
<head>
...
<script src="/hugin.js" type="text/javascript"></script>
...
</head>
```

To interact with the API, first a reference to a HUGIN API (HAPI) instance must be obtained. The HAPI instance provides the logic for performing HTTP communication with the Web service API, caching and reusing the results of idempotent GET requests between each POST request, error handling and other internal service code.

The HDE can then be exercised within any script-element in the document. Here is an example (with minimal error handling code):

```
<script type="text/javascript">
try {
  //create a new HAPI instance
  hapi = new HAPI("/webservice");

  //create a new Domain instance
  domain = hapi.getNewDomain();
```

```
  //create a node in the domain
  A = domain.getNewNode(HAPI.H_CATEGORY_CHANCE,
                        HAPI.H_KIND_DISCRETE,
                        HAPI.H_SUBTYPE_NUMBER);
  A.setName("A");
  A.setNumberOfStates(3);
  A.setStateValue(0, 10);
  A.setStateValue(1, 20);
  A.setStateValue(2, 30);
  ...
} catch (e) {
  alert("An error occurred: " + e.name + "\n" + e.message);
}
</script>
```

The example first creates a HAPI instance. Next it creates a Domain object with a Numbered discrete chance node with name A and three states with values 10, 20 and 30.

3.5 Widgets Library

A widget is a basic visual building block that provides a single point of interaction with application logic. The GUI of an application is constructed by combining several widgets.

The HUGIN Widgets Library is a toolbox of GUI elements for exercising the JavaScript for HUGIN Web Service API through point-and-click on a web page. An infrastructure is provided for updating any widgets related to a particular Domain whenever the state of the Domain changes. This functionality is provided by the WidgetManager, which is used for creating widget instances. A set of basic widgets, each with appropriate hooks for automatic updating and callback to the WidgetManager, are provided:

- TextLabel - *Displays a string of characters.*
- BeliefLabel - *Display the belief for a specific discrete Node state.*
- ExpectedUtilityLabel - *Display the expected utility for a specific discrete Node state, a utility Node or a Domain.*
- FunctionValueLabel - *Display the value of a function Node.*
- ImageLabel - *displays a specific image on the web page based on a custom function.*
- SelectOption - *Presents a number of selectable options.*
- StateSelectOption - *Let the user select a state to enter as evidence for a given discrete Node.*

- TextInput - *presents an editable text field.*
- NumericNodeTextInput - *Presents an editable text field that selects a state of a given discrete numeric Node based on numeric value entered by the user.*
- ContinuousNodeTextInput - *Presents an editable text field for entering user provided value as evidence on a continuous Node.*
- ButtonInput - *Simple button that can perform a custom action when clicked.*
- ButtonInitialize - *Button that performs a Domain.initialize operation when clicked.*
- ButtonPropagate - *Button that performs a Domain.propagate operation when clicked.*
- CustomWidget - *Base class for widgets that can be extended with custom functionality.*

The user may implement Widgets with custom functionality by augmenting a CustomWidget with the needed functionality.

The WidgetManager supports two different methods for including widgets on a web page. A method for simple web applications, where the developer wish to add widgets to the web page as the browser parses the web page HTML source, and a more advanced method where the developer may create and delete widgets at will and inject or remove these widgets at certain places in the web page Document Object Model (DOM).

For simple web applications, the recommended approach to adding HUGIN widgets to the web page is by placing or including the initializing HUGIN code in the web page header, and then adding HUGIN widgets using strategically placed script elements in the web page body. For example:

```
<html>
 <head>
  ...
  <script src="/webservice/hugin.php/hugin.js"
          type="text/javascript"></script>
  <script src="/webservice/hugin.php/hwidgets.js"
          type="text/javascript"></script>
  <script type="text/javascript">
  try {
    // create a new HAPI instance
    hapi = new HAPI("/webservice/hugin.php");

    // load a Domain instance from a .hkb file
    domain = hapi.loadDomain("http://somehost/domain.hkb");

    // create a WidgetManager instance
    // optimize for displaying widgets for nodes A and B
```

```
    widMan = new HuginWidgetManager(domain,
                            { prefetch_by_name: ["A", "B"] });

    // get nodes
    A = domain.getNodeByName("A");
    B = domain.getNodeByName("B");
  } catch (e) {
    alert("An error occurred: " + e.name + "\n" + e.message);
  }
  </script>
  ...
 </head>
 <body>
  ...
  <script type="text/javascript">
  try {
    // add a state selector for selecting state of A
    widMan.addStateSelectOption(A, {});
  } catch (e) {
    alert("An error occurred: " + e.name + "\n" + e.message);
  }
  </script>
  ...
  <script type="text/javascript">
  try {
    // add a belief label for inspecting state 0 of node B
    widMan.addBeliefLabel(B, 0, {});
  } catch (e) {
    alert("An error occurred: " + e.name + "\n" + e.message);
  }
  </script>
  ...
 </body>
</html>
```

For more advanced web applications where the DOM is dynamically modified, and where the simple approach to adding HUGIN widgets is not appropriate, widgets can be injected into the DOM by creating a widget and injecting the corresponding DOM element into the DOM.

In this kind of dynamic web applications where the DOM is subject to many modifications and when widgets may need to be replaced or changed, attention must be paid to making sure that widgets are appropriately entering and leaving the HUGIN widget life cycle as they are added to and removed from the DOM, such that the WidgetManager can keep track of which widgets to update whenever Domain state changes:

```
...
<script type="text/javascript">
try {
  // variables hapi, widMan, B has been properly instantiated
  // as above

  // construct widget for node B
  beliefB = widMan.constructBeliefLabel(B, 0, {});

  // register widget for updates with WidgetManager
  beliefB.addToLifecycle();

  // inject element in DOM
  document.body.getElementById("myid").
          appendChild(beliefB.getDomNode());
} catch (e) {
  alert("An error occurred: " + e.name + "\n" + e.message);
}
</script>
...
```

3.6 An Example using SimpleTrace

Consider the SimpleTrace model from Section 2.2. Figure 2 shows SimpleTrace in HUGIN GUI with entered and propagated evidence $\epsilon = \{\log N_4 = 1.35, \text{Precision} = \text{high}\}$. For an End-User such as a farmer or a veterinarian this user interface may appear highly complex and non-intuitive for the decision making process she or he is facing in an everyday operation. A lack of understanding of the process, the model and the output may reduce the engagement and commitment of the farmer or veterinarian. This again may be fatal to the success of the application. The user interface in Figure 2 is, however, well-suited for interaction with subject matter experts and potential End-Users in a model development phase.

Using the components of the HUGIN Web Service API it is straightforward to develop a user-friendly interface to SimpleTrace. Figure 4 illustrates a simple, efficient and user-friendly front-end to SimpleTrace. The End-User interacts with the underlying model (hosted at a server) using a web page in a standard web browser. This is can serve as a key tool in the model validation process as well as a key element in stakeholder involvement.

Figure 5 shows the front-end in Figure 4 with entered and propagated evidence $\epsilon = \{\log N_4 = 1.35, \text{Precision} = \text{high}\}$. The front-end can be embedded into a larger web-site with an explanation of the information required for the decision making process.

Figure 6 shows how to implement the SimpleTrace front-end shown in Figure 4 and Figure 5. The JavaScript can be embedded into the HTML code or stored in a separate file which is loaded while the page is parsed.

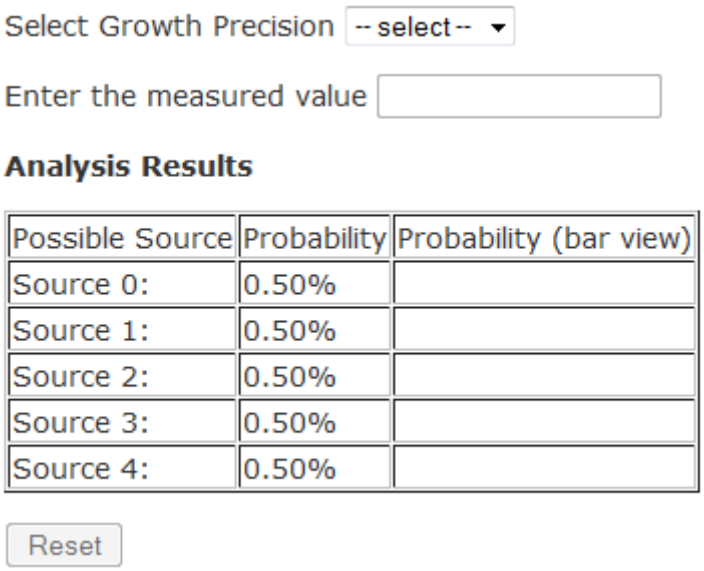

Figure 4: SimpleTrace front-end on the Internet.

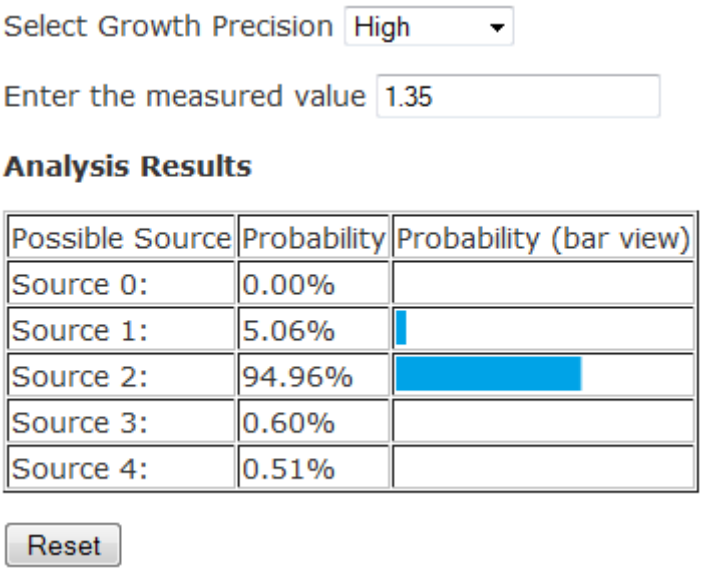

Figure 5: SimpleTrace front-end displaying the result of a belief update with evidence entered.

```
...
hapi = new HAPI("/webservice/hugin.php");
domain =
  hapi.loadDomain("http://solaris/webservice/SimpleTrace2.hkb");
domain.compile ();
widMan = new HuginWidgetManager(domain, {init_compile:false,
     prefetch_by_name: ["S0", "S1", "S2", "S3",
                        "S4", "D_GF", "LogN4" ] });
S0 = domain.getNodeByName("S0");
S1 = domain.getNodeByName("S1");
S2 = domain.getNodeByName("S2");
S3 = domain.getNodeByName("S3");
S4 = domain.getNodeByName("S4");
D_GF = domain.getNodeByName("D_GF");
LogN4 = domain.getNodeByName("LogN4");
...
<p>Select Growth Precision
<script type="text/javascript">
 widMan.addStateSelectOption(D_GF, {retract: [LogN4]});
</script></p>
<p>Enter the measured value
<script type="text/javascript">
 widMan.addNumericNodeTextInput(LogN4, {depends: [D_GF],
                                        deferaction: 1000});
</script>
</p>
...
<tr>
<td>Source 4:  </td>
<td><script type="text/javascript">
  widMan.addBeliefLabel(S4, 1, {display: "percent"});
    </script></td>
<td><script type="text/javascript">
  S4img = imageLabel(widMan, S4, 1, 17);
    </script></td>
</tr>
...
```

Figure 6: HUGIN Web Service code for the SimpleTrace front-end.

The script first loads the PGM from a HUGIN Knowledge Base file, compiles the PGM into a junction tree and assigns identifiers to variables in the model for easy reference. The next steps are to create two input fields and organise the output (both a value and an image representing a belief bar) into a table. The image label is created using the function defined in Figure 7. Notice that the width of the image label is controlled in the update function for the object. The width of the image is controlled by the posterior belief of the specified state of the specified node, e.g., in the example $P(\text{Source } 4 = true \,|\, \epsilon)$.

```
function imageLabel (widMan,  node, state, height) {
  var l  = widMan.addImageLabel( { images: [
              "../data/_uploaded/image/white_bar.png",
              "../data/_uploaded/image/blue_bar.png"],
                                    showimg: 0} );
  l.setUpdateFunction(function() {
                        l.showImage (1);
                        l.setWidth (node.getBelief (state) * 100);
                        l.setHeight (height);
                      });

  return l;
}
```

Figure 7: Function to create an image label where the width of the image is changed in the update function.

It is clear from the example that the Web service API is an efficient and effective tool for deployment of even complex PGMs on the Internet enabling End-Users and other stakeholders to interact with the model using a presentation that is familiar to them.

4 Deployment Examples

This section describes two PGMs for decision support under uncertainty in the domain of food safety. The first example describes a system based on a PGM under development in the CamVac project[2] while the second example describes an operational model for biotracing developed in the BIOTRACER project[3].

4.1 Campylobacter Vaccination in Poultry

Human campylobacteriosis represents an important public health problem. Poultry (consumption and handling) has been identified as one of the main risk factors associated with

[2]CamVac is a project partly funded by the Danish Strategic Research Council (contract 09-067131).
[3]BIOTRACER is a project partly funded by the European Commission through the 6th Framework Programme (contract FP6-2006-FOOD-036272).

human campylobacteriosis cases [6, 38]. In fact, seasonality effects have been detected regarding human campylobacteriosis cases and Campylobacter numbers in chickens [42]. Campylobacter spreads fast within broiler flocks, once a bird has been colonized by Campylobacter, the rest of the birds in the same house will be contaminated within one week [18]. Campylobacter colonizes the chicken intestine, multiplies in the intestinal mucus layer being able to re-invade epithelial cells [50]. Therefore, broilers might carry Campylobacter in higher numbers, even exceeding 10^7 CFU/g of caecal content [45]. In fact, the colonization level can be as high as 10^{10} CFU/g of faeces [47, 46, 32]. Campylobacter originating from faeces of infected chickens will contaminate the food processing environment and will directly affect other carcasses.

The main objective of the CamVac project is to develop a cost-effective vaccination strategy against Campylobacter in poultry, hereby reducing the numbers of Campylobacter in poultry production systems, for instance, commercial broiler flocks. The project aims to identify a vaccination strategy based on reduction, since risk assessment studies have shown that a two log reduction of colonization in poultry can reduce the risk of human infection by 30 times [44]. Campylobacter control strategies can be implemented at different levels of the food chain. In this paper the focus is on the development of a system to support decision making on Campylobacter vaccination of commercial broiler flocks.

In chickens, the immune response against Campylobacter is generally moderate and the absence of a strong immune response has been identified as one of the main challenges for vaccine efficacy to control Campylobacter in chickens [9]. Campylobacter is rarely detected in commercial flocks with birds younger than two weeks of age [3, 49]. It has been suggested that this two weeks *window* could be used strategically to introduce vaccination programs [43]. Therefore, the decision about vaccination needs to be made usually before Campylobacter is introduced in the flock. In fact, there is uncertainty regarding the introduction of Campylobacter into the flock that needs to be taken into account in the decision making process.

Figure 8 shows the structure of a PGM for the campylobacter vaccination at two weeks for commercial broilers. This PGM referred to as ComBVac is an instantiation of a more general PGM referred to as SimpleVac. SimpleVac is similar to the SimpleTrace model, but developed for vaccination decisions at two weeks. The PGM in Figure 8 is being used as an example in the process of developing a decision support tool for a set of vaccine candidates under investigation and development. ComBVac is independent of the properties of a specific vaccine and it has been quantified based on subject matter expert knowledge and information from the literature. The actual quantification of ComBVac is outside the scope of this paper.

Strict on-farm biosecurity can contribute to avoid or reduce Campylobacter colonisation of chickens, in particular, restricting the access of pests, e.g., rodents and flies, into chicken houses will protect against Campylobacter colonization of chickens [15, 36]. Thinning (a depopulation practice consisting on removing a number of birds from the flock) has been identified as a significant risk factor for the introduction of Campylobacter into chicken houses [16, 1].

It is important to distinguish between the true numbers of Campylobacter in poultry

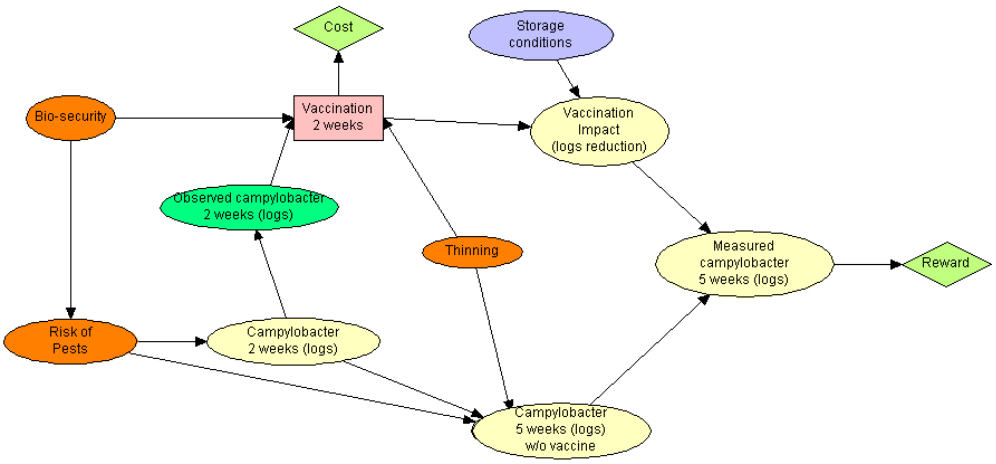

Figure 8: PGM for campylobacter vaccination decision at two weeks for commercial broiler.

flocks and the detected or measured numbers. There are several microbiological techniques available for the detection and enumeration of *Campylobacter spp.* from different sample matrices. However, some techniques are still under development and the detection limit of most methodologies is 100 CFU/g (depending on sample preparation). Therefore, a negative result might actually indicate very low numbers of Campylobacter (1 to 100 CFU).

Figure 9 shows the web interface for the commercial broiler model ComBVac. The web interface is designed to engage the farmers, veterinarians and other stakeholders in the design and development of the system for knowledge management and decision support on vaccination. [48] defines Knowledge Management(KM) as *the set of processes, technology and behaviours that deliver the right content to the right people at the right time and in the right context* so that they can make the best decisions and solve problems. The main aim of KM is to enhance knowledge processing to produce better decisions [13]. Knowledge is the product of complex and multifaceted processes. Furthermore, the creation of knowledge is critical for KM [51].

The interaction between information, technology and people's knowledge is crucial for innovation [5]. Newer technologies such as this web interface are more flexible in relation to supporting individuals' creativity and innovation.

The use of Information and Communication Technologies (ICTs) and selective representation in animal production and food chain management has become vital for sustainable agricultural management and disease control strategies. Selective representation of important data and information is necessary for the efficient use of ICTs in the animal and food industries. The use of ICTs in these industries could be improved in different ways including providing food producers with important information and supporting critical decision making regarding their business. However, human beings are restricted by bounded rationality with a limited capacity to understand the complex world they

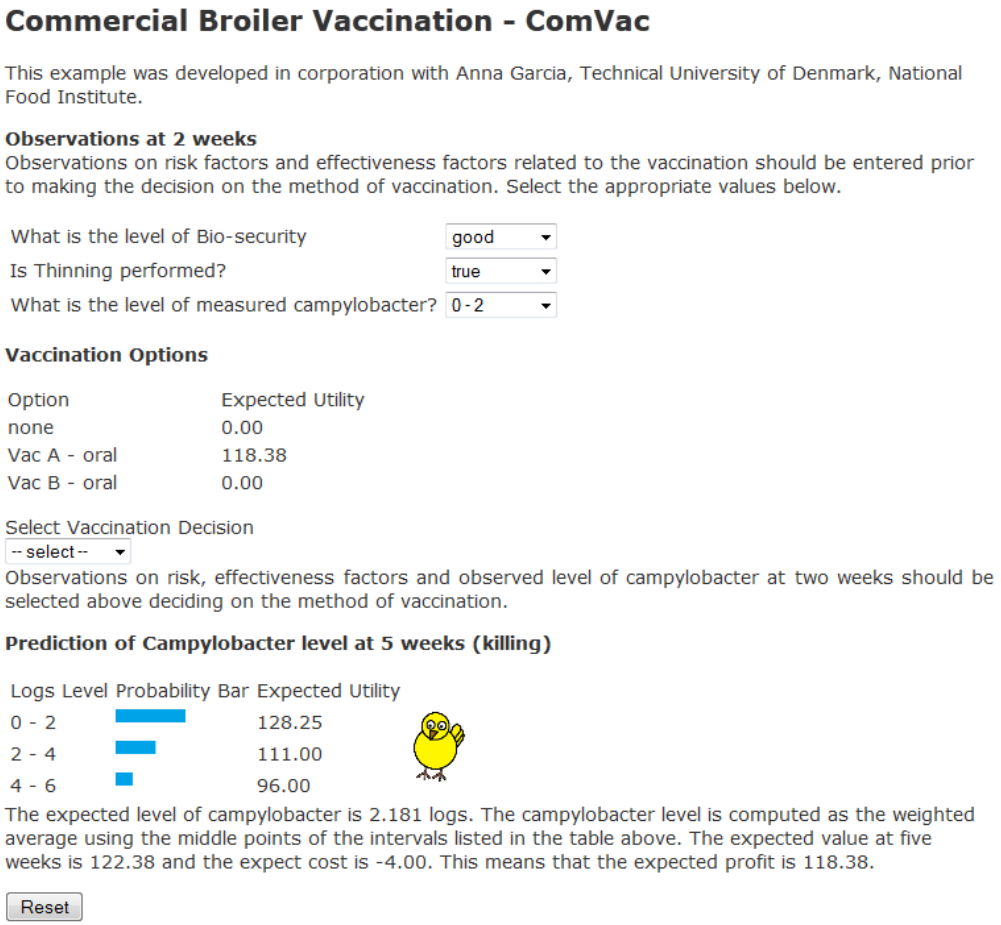

Figure 9: Web interface for the commercial broiler PGM ComBVac.

face. Representing the world originates and also translates on the inability of reaching the real world. Fortunately, humans can be prisoners of their own logic but also capable of critical attitudes [14]. However, the use of ICTs and selective objectification might be completely necessary in some cases when the complexity of the reality we try to represent is very high. Mathematical models such as PGMs and the use of complex statistical programs are a good example.

The web interfaces to ComBVac and the SimpleVac models are important aids in the model development process. These web interfaces have increased and ensured the involvement of stakeholders in the model development process and they supported the elicitation of subject matter expert knowledge by demonstrating the capabilities of the technology and by illustrating what information and data are required for the quantification of ComBVac.

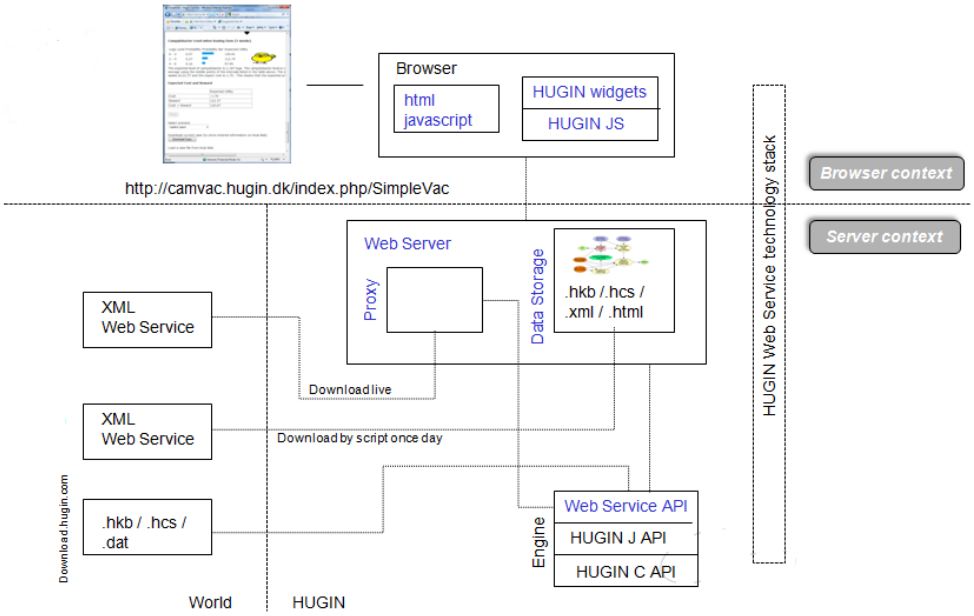

Figure 10: The architecture supporting the web interface in Figure 9.

There are other risk factors and socio-economic variables that could be considered increasing the complexity of the models. However, the final aim of this PGM is for poultry managers to use the model as a tool for decision making on vaccination strategies under conditions of uncertainty. Therefore, we need to balance model efficiency with simplicity and usefulness for the tool to fulfil its purpose.

Figure 10 illustrates how the Web Service architecture supports the web interface in Figure 9. The solution has a server and browser context. In the browser context HTML code and JavaScript are parsed and executed, respectively, and HUGIN widgets are used to present information to the user and received inputs that are communicated to the server context. In the server context, the web server receives and executes commands

using the Web service API. The data storage contains, for instance, a HUGIN Knowledge Base representation of ComBVac. This file is loaded when the page is loaded in the browser context and the ComBVac model is compiled. In the server context, the Web Server and the Engine can interact with other web services and servers as illustrated in the figure.

4.2 Bio-Tracing in the MilkChain Model

BIOTRACER has brought together experts from microbiology, software development, risk assessment, legislation and standards, as well as feed and food manufacturers to develop, among other objectives, novel frontier technologies and exploit them to trace microorganisms and their toxins in selected food chains. In this section, we consider the deployment of a risk assessment model for enterotoxigenic Staphylococcus aureus in milk sold as pasteurized in the United Kingdom [4].

The PGM represents the coupled dynamics of bacteria, toxins and enzymes along the milk production chain. The model includes an explicit identification of contamination sources that include poor farm storage, incomplete pasteurization and post process contamination. The domain integrates many information supplies to support inference concerning the origins of contamination.

The PGM, in this paper referred to as MilkChain, developed for risk assessment on enterotoxigenic Staphylococcus aureus in milk sold as pasteurized is of considerable complexity. MilkChain has a total of ninety variables and the optimal (where the optimality criterion is total state space size) junction tree structure has a total clique state space size $\sum_{C \in \mathcal{C}} ||C|| = 89,056,292$. In the HUGIN Decision Engine each number in a clique table is represented using the data type *double* requiring eight bytes. This means that the representation of the clique tables alone requires approximately 712 MB of RAM. In the server context, a separate domain is created each time the page is loaded in a browser context. This means that a significant amount of RAM is required to support the MilkChain model.

The complexity of MilkChain can be overwhelming to stakeholders such as farmers, or veterinarians, so that an intuitive and efficient interface is essential. The interface can be targeted at specific stakeholder groups to provide resources and facilities that match with particular requirements.

Figure 11 shows a simple, efficient and user-friendly front-end to MilkChain. The End-User interacts with the underlying model (hosted at a server) using a web page in a standard web browser. The End-User selects the appropriate inputs and presses the *Compute* button to propagate evidence in the model and, subsequently, to compute the likelihood ratios.

The deployment of MilkChain is supported by server context PHP code implementing the calculation of the likelihood ratios. This reduces the number of messages passed between the browser and server contexts and as a result increases the time efficiency significantly.

Figure 12 illustrates a special purpose front-end for the MilkChain model. This front-end is developed for mobile devices with a small screen such as, for instance, a smart-

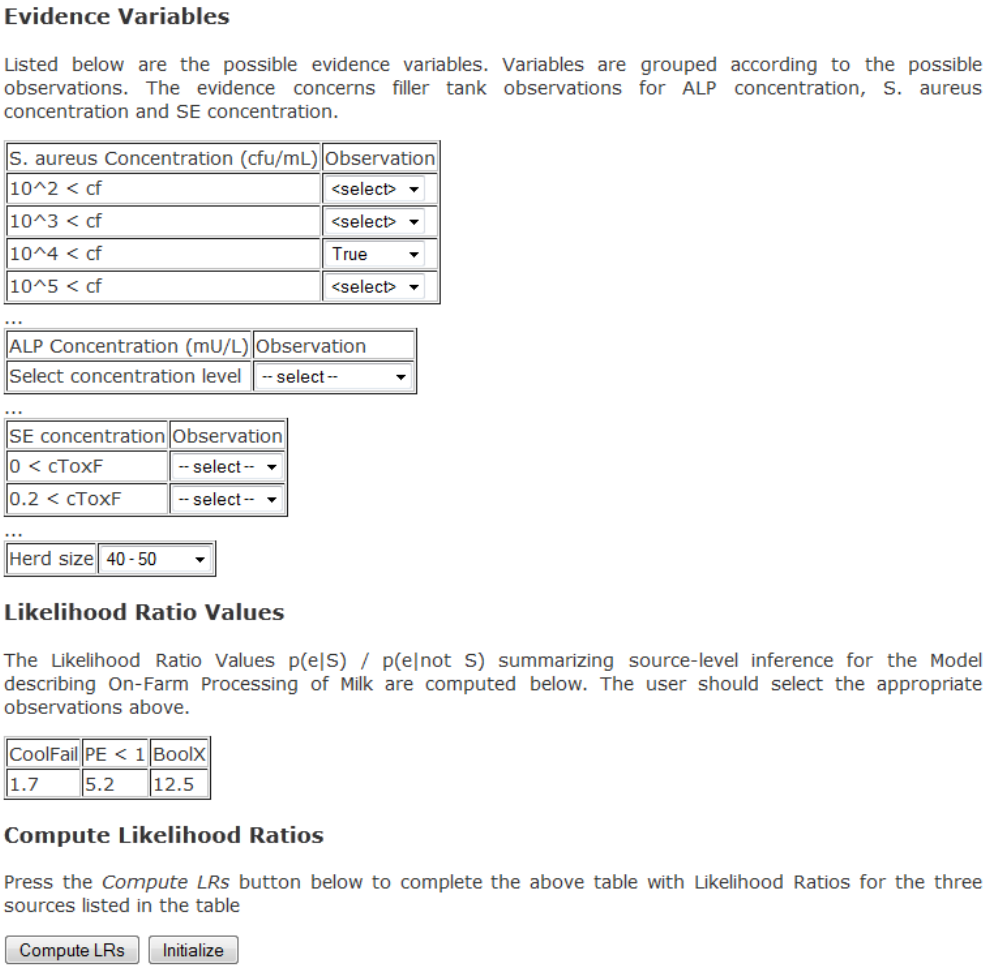

Figure 11: Web interface for bio-tracing in the MilkChain model.

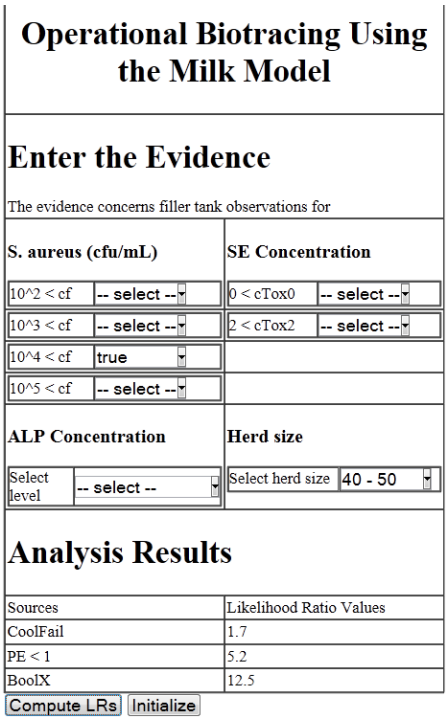

Figure 12: Mobile device interface to the MilkChain model.

phone. The calculations in the server context are performed using the same code as the front-end displayed in Figure 11.

The MilkChain model supports source-level inference for on-farm milk processing. In such a forensic situation, the likelihood ratio $P(\epsilon \mid s)/P(\epsilon \mid \neg s)$, where ϵ is the evidence and s is a source, is used as a more reliable evaluation of the evidence as compared to using posterior probabilities $P(s \mid \epsilon)$. The likelihood ratio (or value of evidence) evaluates the evidence ϵ for a source s relative to an alternate $\neg s$ that expresses the explicit absence of s. These calculations are performed in the server context using dedicated PHP code.

The structure of the server context PHP code to support efficient computation of the likelihood ratio for each potential source is shown below:

```
...
require_once('../webservice/WebService.php');
...
WebService::POST("$host$domain/propagate", "=sum&=normal");
$p_e = (double)
WebService::GET("$host$domain/getNormalizationConstant");
...
$p_CoolFail_given_e = (double)
WebService::GET("$host$CoolFail/getBelief?=1");
...
WebService::POST("$host$CoolFail/selectState", "=1");
WebService::POST("$host$domain/propagate", "=sum&=normal");
$p_CoolFail_and_e = (double)
WebService::GET("$host$domain/getNormalizationConstant");
WebService::POST("$host$CoolFail/retractFindings", "");
...
```

The purpose of the server context PHP code is to improve efficiency by reducing the number of communications between the client browser and the server.

Development of the MilkChain model illustrates several elements in the process for transferring complex systems problems into accessible versatile knowledge bases. Whilst most milk chain stakeholders identify process failures and post process events as their major concerns in relation to safety most cannot, immediately, integrate information concerning microbial populations, process controls, enzyme kinetics and in-line monitors into a single framework that addresses these issues consistently. The MilkChain model development allows subject matter experts to contribute knowledge independently whilst adding to an integrated, and consistent, global picture of milk chain safety. The MilkChain development shows that the total complexity associated with many food chain scenarios is not a barrier to stakeholder involvement in the development of decision support tools. The MilkChain model concerns hazards associated with S. aureus, and its associated enterotoxins, but it is clear that the development includes many elements, associated with food chain operations that are independent of this particular pathogen. In this way the PGM development includes a strong element of transferability, which is

apparent to many stakeholders, that promotes belief in rapid and efficient progressions associated with the knowledge base approach. One area, alternative markers for process control, could easily form an extension of the MilkChain model and could be driven by stakeholders who were not part of the initial development.

The MilkChain model is constructed from expertise that is distributed across the chain i.e. microbiologists add information concerning population dynamics whilst engineers contribute information in relation to pasteurizer settings and controls. In many complex multi-stakeholder food chains this integration of causal beliefs, which is facilitated by a modeler, is the primary construction process. However in some situations, particularly where a chain has a single dominant expertise or information supply, causal construction is not apparent. In these situations a PGM can be constructed, systematically, from a database of observed cases and the development of an interface, and tools supporting decision making, follow. A model, developed alongside the MilkChain model, relating to S. aureus hazards in raw milk cheeses fits this pattern. Raw milk cheese manufacture is usually small scale and often involves only a single local production line. Process information, such as bacterial counts and pH measurements, are collected at strategic points during the cheese history and are used to maintain safety but can also form a database for generating a PGM representing the local cheese making process. New initiatives in food safety, such as biotracing, are most likely to be driven by normative information and the MilkChain example illustrates that PGMs, and appropriate tools that bring these to life on browser pages and on hand held devices, serve this purpose very well for a wide range of end users.

5 Discussion and Concluding Remarks

This paper describes a new architecture – the HUGIN Web Service API – for deployment of PGMs. Efficient deployment of fully specified PGMs was a main design criteria in the development of the HUGIN Web Service API, i.e., the assumption is that the PGM is specified off-line using the HUGIN GUI and deployed using the infrastructure. The fully specified PGM is then deployed, for instance, on the Internet using predefined widgets of the HUGIN Web Service API.

Due to the intuitive graphical nature of PGMs, they provide a platform for constructive discussions on properties of a problem domain. The structure of a PGM facilitates the knowledge engineering process and supports discussions between stakeholders and subject matter experts on properties of the model. The visual representation of relationships between entities of the problem domain enables the involvement of stakeholders at different levels and non-experts. The infrastructure presented in this papers adds significant value to this process by enabling users to interact with the PGM through a custom interface. It enables and supports stakeholder involvement through the possibility to interact with prototype models, which helps to increase the understanding and appreciation of models as users can interact with models on their own and at their convenience. The ability to interact with the underlying PGM is according to our experience highly valuable to non-experts in PGMs.

Our experience from developing and using the architecture in the BIOTRACER and CamVac projects as well as other projects and applications show that the ability to demonstrate the power of PGMs using a web-based front-end can help to build commitment and engagement from partners not directly involved in the process of constructing the PGM. A common comment from subject matter experts is that modeling domains as PGMs is a different way of thinking for them. A front-end enabling users to interact with an underlying PGM can assist in the learning process and in buidling the skills to support an efficient knowledge elicitation process.

In projects such as BIOTRACER and CamVac where partners are distributed geographically and with limited opportunities for meeting in person, the architecture has been used as a platform for engaging partners early in the development process as well as for discussing and making clear the aim and purpose of the knowledge engineering process to subject matter experts. It is more easy for other partners to relate to the front-end and appreciate the potential of PGMs than to understanding the details of the technology or being faced with complex models. By involving stakeholders and subject matter experts early on in the model development process a high engagement and commitment and a high sense of involvement and ownership can be achieved.

Section 3.5 presents a library of widgets developed to make it easy and efficient to create a web-page interface to a PGM. This library of predefined widgets and a flexible programming interface enable the user to construct special-purpose and tailored interfaces for a specific PGM. The system development process has followed agile principles for software development involving end-users in the process helping to ensure that the requirements of the users are met to a high degree. The support for computing likelihood ratios for the MilkChain is an example of a special-purpose and tailored interface. These types of custom extensions are not possible with existing COTS software packages for PGMs.

Furthermore, as mentioned above, the complexity of large and technical PGMs such as MilkChain can be overwhelming to stakeholders such as farmers, or veterinarians. This means that an intuitive and efficient interface is essential and it should be targeted at specific stakeholder groups to provide resources and facilities that match with particular requirements. The infrastructure is designed to meet this need.

The HUGIN Web Service API enables fast, efficient and effective deployment of PGMs as explained and illustrated by the examples of this paper. The architecture has been used to deploy models and examples from not only the domain of food safety, but in a range of other domains including medicine, insurance and banking. Being able to deploy PGMs efficiently and effectively as a web service makes it easier to demonstrate the potential of PGMs to decision makers. It is an important tool for prototyping interfaces to (complex) PGMs as well as for educational and commercial purposes.

Using the HUGIN Web Service API it is efficient and effective to target specific elements of a complex domain to appropriate stakeholder groups. The architecture can be deployed on a local desktop computer, on intranets and the Internet. A PGM delivered as a web service reduces the need to change existing systems or processes.

The infrastructure has been very well received by users and it is very popular among existing and new partners. We have experienced a high interest from users of BBN

software to deploy models using the architecture. A number of web solutions using the architecture to deliver PGMs for decision support are planned and under development. This paper describes two use-cases where the architecture has been used to provide easy-to-use user interfaces to both relatively simple and highly complex PGMs.

A web-site can be an efficient and effective communication channel and an important element in, for instance, dissemination efforts related to research and development projects as well as for demonstrating the capabilities of solutions based on PGMs.

The example in Section 3.5 illustrates the programming efforts it takes to create a web interface to a PGM using widgets for selecting states and showing updated beliefs of nodes in the PGM. This JavaScript example illustrate the simplicity of the interface with respect to deployment of models. Once the server is running, it is straightforward to develop interfaces for different PGMs. It is a matter of few hours or even less than one hour of work to design and implement an interface for a model like SimpleTrace for more complex models like MilkChain, it is necessary to make customizations using the APIs to meet the requirements not supported directly by elements of the widgets library. This demonstrates the flexibility of the architecture.

6 Future Work

The HUGIN Web Service API includes a library of default widgets for easy deployment of PGMs. This includes widgets for entering evidence for different variable types and for displaying the results of inference such as posterior beliefs and expected utilities. The widgets are useful for fast and efficient deployment of PGMs on the Internet, for instance, for demonstration purposes. Future work includes developing additional widgets, in particular, for model construction, revision and adaptation as well as widgets for interfacing with other applications. The latter should support seamless integration with third party applications.

The Web service API has been applied to deploy PGMs covering a wide range of problem domains including food safety, agriculture, environmental protection, sports, insurance and finance. Future work includes the development of additional PGMs both for demonstration purposes and for educational as well as commercial purposes.

Acknowledgments

This research and development work was partly funded by the Danish Strategic Research Council (CamVac contract 09-067131) and the European Commission through the 6th Framework Programme (BIOTRACER contract FP6-2006-FOOD-036272).

A free version of the HUGIN software tools for demonstration purposes only can be downloaded from the web-site *www.hugin.com* and example applications of the HUGIN Web Service API can be found at the web-sites *milk.hugin.com* and *camvac.hugin.com*.

References

[1] A. Adkin, E. Hartnett, L. Jordan, D. Newell, and H. Davison. Use of a systematic review to assist the development of Campylobacter control strategies in broilers. *Journal of Applied Microbiology*, 100:306–315, 2006.

[2] S. K. Andersen, K. G. Olesen, F. V. Jensen, and F. Jensen. HUGIN — a Shell for Building Bayesian Belief Universes for Expert Systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1080–1085, 1989.

[3] A. Annan-Prah and M. Janc. The mode of spread of Campylobacter jejuni/coli to broiler flocks. *Journal of Veterinary Medicine*, 35:11–18, 1988.

[4] G. C. Barker and N. Gomez-Tome. A Risk Assessment Model for Enterotoxigenic Staphylococcus aureus in Pasteurized Milk: A Potential Route to Source-Level Inference. Risk Analysis. doi: 10.1111/j.1539-6924.2011.01667.x, 2011.

[5] S. Brelade and C. Harman. *Practical Guide to Knowledge Management.* London, GBR: Thorogood, 2003.

[6] B. Christenson, A. Ringer, C. Blucher, H. Billaudelle, KN. Gundtoft, G. Ericksson, and M. Bottiger. An outbreak of campylobacter enteritis among the staff of a poultry abattoir in Sweden. *Scand J Infect Dis*, 15:167–172, 1983.

[7] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems.* Springer-Verlag, 1999.

[8] A. Darwiche. *Modeling and Reasoning with Bayesian Networks.* Cambridge University Press, 2009.

[9] M.R. de Zoete, J.P. van Putten, and J.A. Wagenaar. Vaccination of chickens against Campylobacter. *Vaccine*, 25:5548–57, 2007.

[10] E. Ejsing, P. Vastrup, and A. L. Madsen. Probability of default for large corporates. In *Bayesian Networks: A Practical Guide to Applications*, chapter 19, pages 329–344. Wiley, New York, 2008.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616, hypertext transfer protocol – http/1.1, 1999.

[12] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2:115–150, May 2002.

[13] J. Firestone and M. McElroy. *Has Knowledge Management Been Done?* Bradford, UK: Emerald Group Publishing Limited, 2005.

[14] A. C. Fors. Being-with information technology. critical explorations beyond use and design. PhD dissertation, 2006. Available online: www.diva-portal.org/diva/getDocument?urn_nbn_se_umu_diva-748-2__fulltext.pdf [Accessed 14 March 2012].

[15] B. Hald, H. Skovgard, K. Pedersen, and H. Bunkenborg. Influxed insects as vectors for Campylobacter jejuni and Campylobacter coli in Danish broiler houses. *Poult Sci*, 87:1428–1434, 2008.

[16] I. Hansson, N. Pudas, B. Harbom, and E.O. Engvall. Within-flock variations of Campylobacter loads in caeca and on carcasses from broilers. *International Journal of Food Microbiology*, 141(1-2):51–55, 2010.

[17] R. A. Howard and J. E. Matheson. Influence diagrams. In *Readings in Decision Analysis*, chapter 38, pages 763–771. Strategic Decisions Group, Menlo Park, CA, 1981.

[18] W.F. Jacobs-Reitsma. Aspects of epidemiology of Campylobacter in poultry. *Vet. Q.*, 19:113–117, 1997.

[19] F. Jensen, U. B. Kjærulff, M. Lang, and A. L. Madsen. HUGIN - The Tool for Bayesian Networks and Influence Diagrams. In *First European Workshop on Probabilistic Graphical Models*, pages 212–221, 2002.

[20] F. V. Jensen and F. Jensen. Optimal junction trees. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 360–366, 1994.

[21] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282, 1990.

[22] F. V. Jensen and T. D. Nielsen. *Bayesian Networks and Decision Graphs*. Springer, 2nd edition, 2007.

[23] Frank Jensen, Finn V. Jensen, and Søren Dittmer. From influence diagrams to junction trees. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 367–373, San Francisco, 1994. Morgan Kaufmann Publishers.

[24] Jetty. The Jetty Project, www.eclipse.org/jetty/, 2012. A Java based web server implementation [Accessed 10 March 2012].

[25] JSLint. JSLint Project, www.jslint.com/lint.html, 2012. Static code analysis tool for JavaScript [Accessed 10 March 2012].

[26] U. B. Kjærulff and A. L. Madsen. *Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis*. Springer, 2008.

[27] D. Koller and N. Friedman. *Probabilistic Graphical Models — Principles and Techniques*. MITPress, 2009.

[28] K. Kristensen and I. A. Rasmussen. The use of a Bayesian network in the design of a decision support system for growing malting barley without use of pesticides. *Computers and Electronics in Agriculture*, 33:192–217, 2002.

[29] S. L. Lauritzen and F. Jensen. Stable local computation with mixed Gaussian distributions. *Statistics and Computing*, 11(2):191–203, 2001.

[30] S. L. Lauritzen and D. Nilsson. Representing and solving decision problems with limited information. *Management Science*, 47:1238–1251, 2001.

[31] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royak Statistical Society, B.*, 50(2):157–224, 1988.

[32] D. Ltticken, R.P.A.M. Segers, and N. Visser. Veterinary vaccines for public health and prevention of viral and bacterial zoonotic diseases. *Rev. sci. tech. Off. int. Epiz.*, 26(1):165–177, 2007.

[33] A. L. Madsen, F. Jensen, U. B. Kjærulff, and M. Lang. Hugin - the tool for bayesian networks and influence diagrams. *International Journal on Artificial Intelligence Tools*, 14(3):507–543, 2005.

[34] A. L. Madsen and F. V. Jensen. Lazy propagation: A junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*, 113(1–2):203–245, 1999.

[35] A. L. Madsen, M. Lang, U. Kjærulff, and F. Jensen. The hugin tool for learning bayesian networks. In *Proceedings of the Seventh European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 549–605, 2003.

[36] F.D. McDowell, S.W.J. andMenzies, S.H. McBride, and A. Oza. Campylobacter spp. in conventional broiler flocks in Northern Ireland:epidemiology and risk factors. *Prev Vet Med*, 84:261–276, 2008.

[37] R. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2003.

[38] J. Neimann, J. Engberg, K. Molbak, and H.C. Wegener. A case-control study of risk factors for sporadic campylobacter infections in Denmark. *Epidemiology and Infection*, 130:353–366, 2003.

[39] D. Nilsson and F. V. Jensen. Probabilities of future decisions. In *Proceedings from the International Conference on Informational Processing and Management of Uncertainty in knowledge-based Systems (IPMU)*, pages 144–1462, 1998.

[40] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Series in Representation and Reasoning. Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[41] O. Pourret, P. Naim, and B. (Eds.) Marcot. *Bayesian Networks: A Practical Guide to Applications*. Wiley, New York, 2008.

[42] F. Reich, V. Atanassova, E. Haunhorst, and G. Klein. The effects of Campylobacter numbers in caeca on the contamination of broiler carcasses with Campylobacter. *Int J Food Microbiol.*, 127(1-2):116–120, 2008.

[43] B.E Rice, D.M. Rollins, E.T. Mallinson, L. Carr, and S.W. Joseph. Campylobacter jejuni in broiler chickens: colonization and humoral immunity following oral vaccination and experimental infection. *Vaccine*, 15(17-18):1922–1932, 1997.

[44] H. Rosenquist, B. Norrung N.L. Nielsen, H.M. Sommer, and B.B. Christensen. Quantitative risk assessment of human campylobacteriosis associated with thermophilic Campylobacter species in chickens. *International Journal of Food Microbiology*, 83:87–103, 2003.

[45] H. Rosenquist, H.M. Sommer, N.L. Nielsen, and B.B. Christensen. The effect of slaughter operations on the contamination of chicken carcasses with thermotolerant Campylobacter. *International Journal of Food Microbiology*, 108:226–232, 2006.

[46] O. Sahin, TY. Morishita, and Q. Zhang. Campylobacter colonization in poultry: sources of infection and modes of transmission. *Animal health Res Rev*, 3:95–105, 2002.

[47] T. Stas, F.T.W. Jordan, and Z. Woldehiwet. Experimental infection of chickens with Campylobacter jejuni: strains differ in their capacity to colonize the intestine. *Avian Pathol*, 28:1999, 61-64.

[48] J. Stemke. Developing an integrated enterprise-wide knowledge architecture. Presentation from the APQC Conference - Next Generation KM, 2001. Available online: eos.gsfc.nasa.gov/eos-ll/references/Developing-Integrated-Architecture.ppt [Accessed 14 March 2012].

[49] N.J. Stern. Reservoirs for Campylobacter jejuni and approaches for intervention in poultry. In *I. Nachamkin, M.J. Blaser and L.S. Tompkins, Editors, Campylobacter jejuni: Current Status and Future Trends, American Society for Microbiology, Washington, DC*, pages 49–60. 1992.

[50] K. Van Deun, F. Pasmans, R. Ducatelle, B. Flahou, K. Vissenberg, A. Martel, W. Van Den Broeck, F. Van Immerseel, and F. Haesebrouck. Colonization strategy of Campylobacter jejuni results in persistent infection of the chicken gut. *Veterinary Microbiology*, 130:285–297, 2008.

[51] N. Wickramasinghe and Dag K. J. E. von Lubitz. *Knowledge-Based Enterprise: Theories and Fundamentals.* Hershey PA: Idea Group Pub., 2007.

[52] Yahoo. Yahoo Developer Network YUI Compressor Project, developer.yahoo.com/yui/compressor/, 2012. A JavaScript processing tool that preserve the operational qualities of the code while reducing its overall byte footprint [Accessed 10 March 2012].